Dynamic scheduling

A major limitation of simple pipelining techniques is that they use in-order instruction issue and execution: instructions are issued in program order and, if an instruction is stalled in the pipeline, no later instructions can proceed. Thus, if there is a dependence between two closely spaced instructions in the pipeline, this will lead to a hazard and a stall will result.

To allow us to continue executing instructions that have no dependences on the current, stalled one, we need to let them run as soon as all their data operands are available. Such a pipeline does *out-of-order execution*, which also implies *out-of-order completion*.

Out-of-order execution, however, introduces the possibility of WAR and WAW hazards, which do not exist in the standard MIPS pipeline with in-order execution.

Out-of-order completion also creates major complications in handling exceptions. Dynamic scheduling with out-of-order completion must preserve exception behaviour in the sense that *exactly* those exceptions that would arise if the program were executed in strict program order *actually* do arise. Dynamically scheduled processors preserve exception behaviour by delaying the notification of an associated exception until the processor knows that the instruction should be the next one completed.

Although exception behaviour must be preserved, dynamically scheduled processors could generate *imprecise* exceptions.

Imprecise exception

An exception is imprecise if the processor state when an exception is raised does not look exactly as if the instructions were executed sequentially in strict program order.

Imprecise exceptions can occur because of two possibilities:

1. The pipeline may have *already completed* instructions that are *later* in program order than the instruction causing an exception.

2. The pipeline may have *not yet completed* some instructions that are *earlier* in program order than the instruction causing the exception.

Imprecise exceptions make it difficult to restart execution after an exception.

To allow out-of-order execution, we essentially split the ID pipe stage of our simple five-stage pipeline into two stages:

1. **Issue**: decode instructions and check for structural hazards.
2. **Read operands**: wait until no data hazards, then read operands.

The scoreboarding approach

See scoreboarding.

Tomasulo's algorithm

See Tomasulo's algorithm.